第7章

CMOSイメージ・センサで読み込んだ画像のエッジ検出を行う

画像処理回路の製作

山際伸一，José Germano

本誌付属FPGA基板の画像処理回路への活用例を紹介する．ここでは，CMOSイメージ・センサから画像を読み込み，読み込んだ画像データに対してエッジ検出処理を行う．処理後の画像は，シリアル通信でパソコンに転送して表示する．エッジ検出アルゴリズムには，Sobelフィルタを用いる．（編集部）

デジタル・カメラや携帯電話のように，イメージ・センサを搭載する小型の機器が増えています．このような携帯機器では，小型化によって受ける制約を，画像処理などの技術で補うことが求められます．光学的に優れたレンズは，どうしても大きくなってしまうためです．ひずみや手ぶれの補正，ディジタル・ズームといった機能を画像処理で実現することは，製品を小型化する上で重要な技術となります．

画像処理は，CPUで動作するソフトウェアで実現することも可能です．しかしCPU処理能力によるフレーム・レートの制約から，カクカク動くと感じられてしまうことがあります．

そこで画像処理アルゴリズムをハードウェアで実装することが欠かせない技術になります．消費電力の点でも，CPUによるソフトウェア処理よりも，特化されたハードウェアのほうが低消費電力を図れます．

本稿では，CMOSイメージ・センサから取り込んだ画像を処理するアプリケーションを紹介します．Spartan-3Eには18ビットの整数乗算器がハード・マクロで内蔵されているため，掛け算を頻繁に使う画像処理などのディジタル信号処理に適します．

図1　画像処理システムの構成
カメラ制御モジュール，画像処理モジュール，データ転送モジュール，表示ソフトウェアから構成される．

1．システム設計

システム全体の構成を**図1**に示します．

- カメラ制御モジュール
- 画像処理モジュール
- データ転送モジュール
- 表示ソフトウェア

から構成されます．

カメラ制御モジュールは，CMOSイメージ・センサからの画素値を読み取ります．その画素値は，画像処理モジュールに転送され，画像処理を行います．画像処理後のデータは，データ転送モジュールによりFPGA外部に送出し，パソコンの表示ソフトウェアで表示します．

今回は，画像処理モジュールでエッジ検出処理を施して

KeyWord　FPGA，CMOSイメージ・センサ，画像処理，エッジ検出，Sobelフィルタ，OV9650，SCCB，I2C-bus，Open Cores，WISHBONE

みます.

● 設計仕様を決める

ここで設計仕様として，以下の事柄を考慮します.

(1) 輝度を画素値として用いる

今回は，画素値として輝度を使用します．従って，白黒画像を扱うことになります．CMOSイメージ・センサは，8ビットの輝度データを出力します.

フルカラー画像を扱いたい場合は，RGBのそれぞれの画素値に対して同じ処理を個別に行えば可能です.

(2) 画像サイズ

今回は，128×64ピクセルの画像を扱います．画素値は8ビットですから，8Kバイトの画像データをハードウェアで扱うことになります.

(3) できるだけデータをバッファリングしない

近年のFPGAは多くのメモリを搭載していますが，画像処理アプリケーションでは十分とは言えません．付属FPGA基板に搭載されているXC3S250Eのメモリ・ブロックは約192Kビット（約24Kバイト，パリティを除く）しかありません．従って，QVGA（320×240ピクセル）の画像すら扱えないことになります.

今回は，1フレーム分の画素データではなく，画像処理モジュールにおける計算で必要な最低限度の画素データだけをバッファリングします．入力されてきたデータに対して順次計算を行って，その結果を次々とデータ転送モジュールから出力します.

128×64ピクセルの画像の場合，すべての画像をバッファリングすることも可能ですが，大きな画像を扱うことが可能なように，すべてのモジュールにバッファを置かないようにします．しかし，データ転送モジュールでは表示ソフトウェアの性能に左右されてしまうので，バッファを置くようにします.

● エッジ検出アルゴリズムはSobelフィルタ

画像処理モジュールに実装するエッジ検出アルゴリズムを考えます.

画像処理では，ピクセルの値を色の濃さとして捉えるため，「濃度値」とも言います．エッジ検出処理では，この濃度値に高低差を大きく付けることになります．つまり，あるピクセルの濃度値が，周囲のピクセルの濃度値と比べて変化が大きいときには，そのピクセルの濃度値をより大きくします．反対に変化が小さい場合には，より小さくする計算をします．あるピクセルを中心としてその周辺の濃度値から新たな濃度値を求めるので，畳み込み演算（コンボリューション）とも呼ばれます.

エッジ検出には，Sobelフィルタ，Prewittフィルタ，2次元FIR（finite impulse response）フィルタ，2次元IIR（infinite impulse response）フィルタなど，多くアルゴリズムが提案されています．ここでは，2次元のエッジ検出とノイズ耐性の良さなどの理由から，頻繁に用いられる方法の一つであるSobel（ソベル）フィルタを用います.

Sobelフィルタは，あるピクセルの周辺ピクセルの濃度の微分を求め，それを新たな濃度値とします．つまり，周辺のピクセルとの濃度値の差分を加算することで，計算対象のピクセルの濃度値を強弱させます．この差分は2次元に対して行われますので，垂直方向の係数行列Vと水平方向の係数行列Hのそれぞれを，対象となるピクセルとその周辺からなる行列Iの対応する各成分を掛け算し，足し合わせた値の絶対値になります．この演算の数式と概念を**図2**にします.

図2の数式から分かるように，Sobelフィルタをハードウェアに実装するためには，掛け算回路が多数必要になります．この計算をFPGAに実装することを考えると，論理ブロックで掛け算回路を実現するのではなく，Spartan-3Eのようにハード・マクロの乗算器を持つFPGAが理想的といえます.

$$I = \begin{bmatrix} P_0 & P_1 & P_2 \\ P_3 & P_4 & P_5 \\ P_6 & P_7 & P_8 \end{bmatrix}$$

$$P' = \left| \sum_{i=0}^{2} \sum_{j=0}^{2} I_{ij} V_{ij} \right| + \left| \sum_{i=0}^{2} \sum_{j=0}^{2} I_{ij} V_{ij} \right|$$

$$V = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

上の計算では，下のような3×3画素の中心における
グラディエント（↑）を求めているとイメージできる.

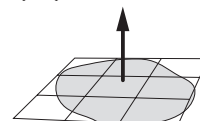



図2 Sobelフィルタ

ピクセルP_4とその周辺のピクセルからなる3×3の行列Iから，垂直方向の係数行列Vと水平方向の係数行列Hを用いてP_4のエッジを強調した濃度値P'を求める．I_{ij}，V_{ij}，H_{ij}は，それぞれ行列I，V，Hの各要素を示す.

● **データ転送プロトコルを決める**

データ転送モジュールは，処理後の画像データが送り出します．この際，勝手な順序でデータを送り出していては，受け取る側（表示ソフトウェア）で正しく情報を受け取ることができません．画像サイズや表示する順番のほか，エラーなく通信されているかを確認する手段などについて，送信側と受信側で決まりを作っておく必要があります．データ転送モジュールは，**図3**に示すようなフォーマットで処理した画像を送り出すようにします．

(1) ヘッダ情報

データ転送モジュールは，最初に0xFF，0x00，0xFF，0x00という4バイト分のヘッダ情報を送出します．これにより，表示ソフトウェアは「画像が取り込まれた」ということを認識します．

(2) 画像情報

ヘッダ情報に続く8バイトは，画像の幅と高さをピクセル単位で表します．表示ソフトウェアはこの情報から，ピクセル・データのデータ量を計算します．

(3) ピクセル・データ

ピクセル・データは，画像の左上のピクセルから右下のピクセルへと送出していきます．画像の幅が分かっているので，表示ソフトウェアによって画像として正しく表示できます．

(4) フッタ情報

データ転送モジュールは，すべてのピクセル・データを送出したら，最後にフッタ情報として0x00，0xFF，0x00，0xFFという4バイトのデータを送出します．表示ソフトウェアでは，このフッタ情報がこない場合は，通信が正しく行われていないかシステム上の問題が発生していると判断してエラーとします．

上記の通信プロトコルは，簡易的なものです．外来ノイズなどの影響で，ヘッダのバイト列とフッタのバイト列が現れることが，ごく小さい可能性ながらありえるためです．より頑丈なプロトコルにするためには，データ転送モジュールと表示ソフトウェアとの間でコマンドなどを設けて対話させるようにします．

図3　データ・フォーマット
データ転送モジュールから表示ソフトウェアにデータを送るときのフォーマットである．

2．回路の設計と製作

FPGAに実装する回路の構成を**図4**に示します．このほかに画像を取り込んだり，処理したデータをパソコンに転送するための外部回路が必要になります．

● **周辺回路を製作する**

図5に付属FPGA基板に対して拡張する回路を示します．

付属FPGA基板上には，オプションのスイッチとクロック発振器を実装します．このスイッチはリセットとして使用します．クロック発振器は，今回は50MHzを使いました．これはシステム・クロックとして使います．

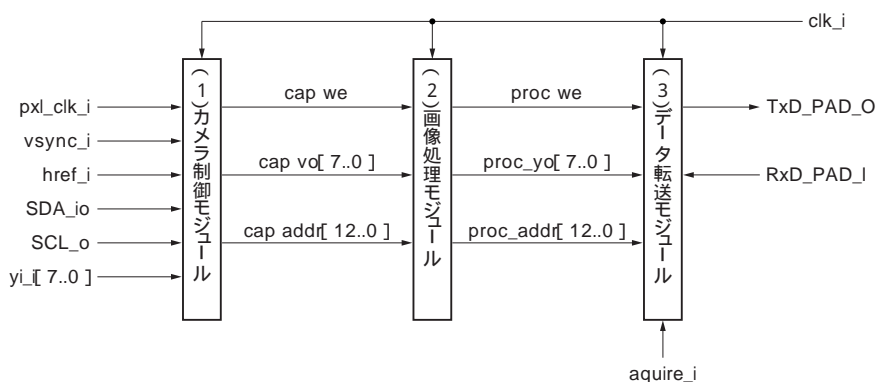
図4
FPGAに実装する回路の構成
FPGAには，カメラ制御モジュール，画像処理モジュール，データ転送モジュールの三つのモジュールを搭載する．このほかに画像を取り込んだり，処理したデータをパソコンに転送するための外部回路が必要になる．

付属基板の外部には，CMOSイメージ・センサを接続します．今回は，米国OmniVision Technologies社の「OV9650」を使用しました(実際には光学系が付属したモジュールを利用)．このほか，画像取り込み用のスイッチとパソコンへのデータ転送をシリアル通信(RS-232-C)で行うためのRS-232-Cトランシーバが必要です．

写真1に周辺回路を実装した基板の外観を示します．

● カメラ制御モジュールでセンサのタイミングを管理

カメラ制御モジュールは，CMOSイメージ・センサ・モジュールが出力する同期信号のVSync，Hsyncと，ピクセル・クロックと呼ばれるピクセル値の出力同期信号を読み取り，8ビットの輝度データを出力します．

多くのCMOSイメージ・センサは，クロック信号を与えることにより，ピクセル・クロックを出力します．今回の場合，クロック周波数の50MHzを16分周したクロックをEXCLKに入力すると，それをさらに32分周したクロック［50MHz/(16×32)］がピクセル・クロックとして出力されてきます．

各ピクセル値は，ピクセル・クロックの立ち上がりエッジで有効な値が保持されています．しかし，ピクセル・クロックの立ち上がりエッジであっても，VSyncとHSyncを連携させないと，正しいピクセル値が得られません．

CMOSイメージ・センサにおける有効なピクセル値が出力されるタイミングを，**図6**に示します．VSyncとVSyncの間が1画像(フレーム)です．一つのフレームは複数の行(ライン)からなります．ラインはHSyncとHSyncに挟まれたところに現れます．HSyncから有効なピクセルが現れるまでは，CMOSイメージ・センサの特性によって異なる時間の遅延があります．その遅延分だけ待った後に有効なピクセル値が現れます．有効なデータが得られている間だけ，ピクセル・クロックの立ち上がりで取得します．従って，VSyncがアサートされ，そのあとにHSyncが立ち下がってから，遅延分時間をカウンタで計り，タイミングを合わせてピクセル値を取得します．

ところで，今回使用したCMOSイメージ・センサ・モジュールには，HREFという便利な信号が用意されています．HREF信号は，HSyncからの一定時間の遅延の後，有効なピクセル値が出力されるタイミングでアサートされるイネーブル信号です．従って，VSyncのアサートの後，フレームが開始されるので，HREFがアサートされている間，

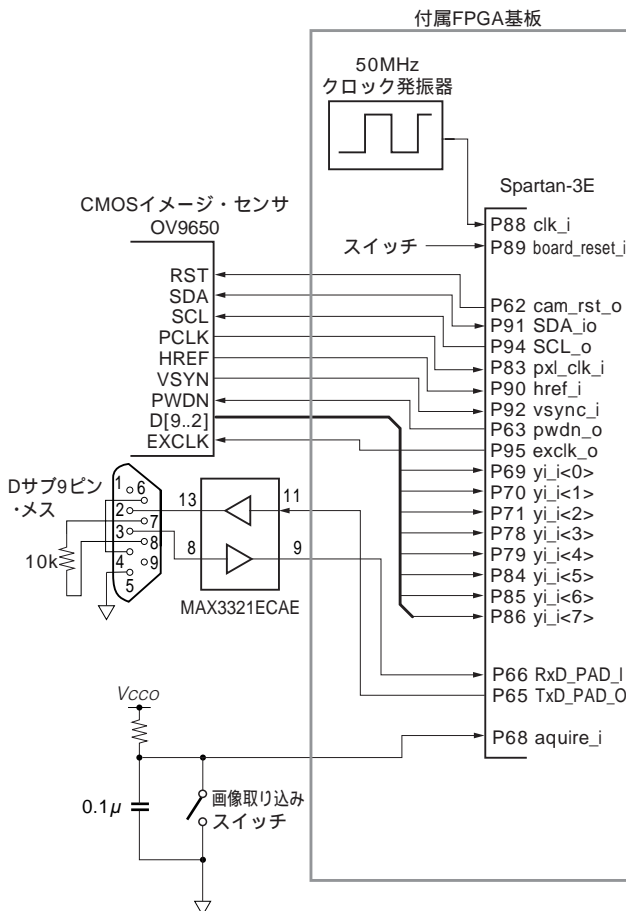



図5　画像処理システムの回路
付属FPGA基板上には，オプションのスイッチとクロック発振器を実装する．付属基板の外部には，CMOSイメージ・センサ，画像取り込み用のスイッチ，RS-232-Cトランシーバを接続する．

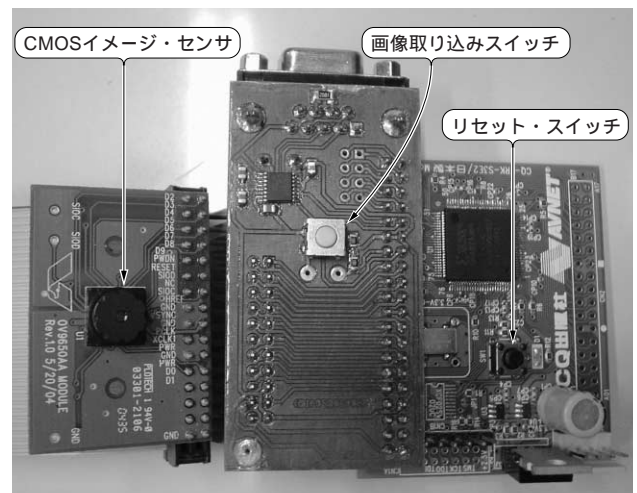
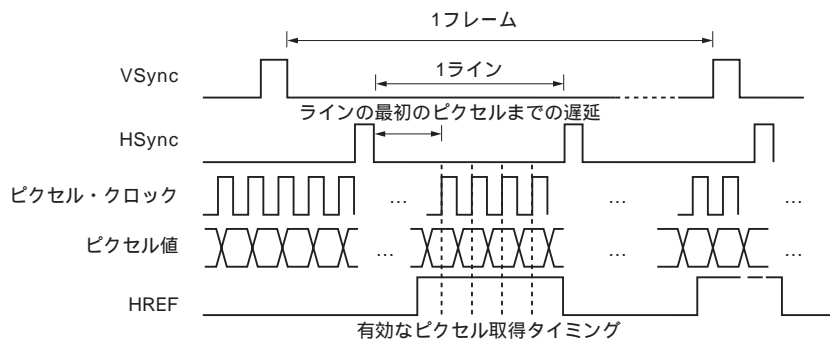



写真1　製作した回路の外観
左側にあるのがCMOSイメージ・センサ．光学系が付属した基板を利用した．中央が画像取り込み用のスイッチとRS-232-Cトランシーバである．

図6
ピクセル取得のタイミング

ラインを読み取る回路を作ればよいことになります．

また，カメラ制御モジュールでは，起動時（リセット解除後），CMOSイメージ・センサ・モジュールの設定を行います．今回は，モジュール開発元のOmniVision社のSCCB（Serial Camera Control Bus）インターフェースを実装することになります．SCCBはI^2C-bus（p.92のコラム「I^2C-bus」を参照）に似ているインターフェースで，1バス・サイクルで2バイトのデータを転送します．I^2Cと異なるのはACKビットの位置がドント・ケア・ビットとして定義されているところです．SCCBインターフェースでカメラ内部のレジスタを書き換え，カメラの解像度などを設定します．

● 画像処理モジュールは汎用性を重視

画像処理モジュールは，汎用性を考えてインターフェース仕様を決定しました．モジュールの入出力仕様を維持したまま内部論理を変更することで，さまざまなアルゴリズムを利用できるように考えました．

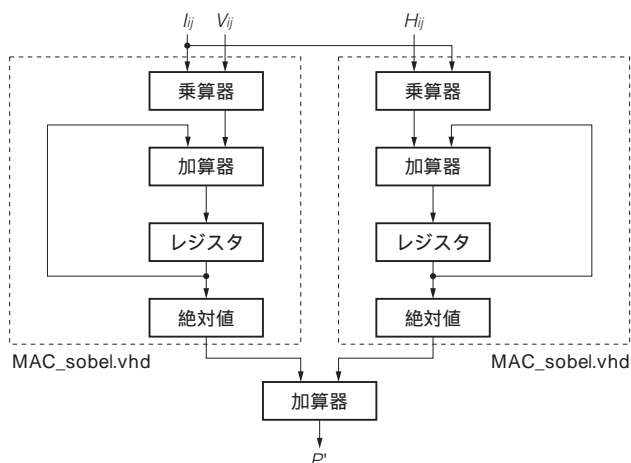



図7　Sobelフィルタ演算の回路構成
繰り返し計算で実現している．

今回は，Sobelフィルタによるエッジ検出回路を実装しています．前述のとおり，Sobelフィルタは入力ピクセルとV行列，H行列の各要素の積和演算で行われます．これは18個の乗算器とそれらを同時に足す加算器で実装可能です．

しかし，ビット数が多すぎる加算器は処理に時間を要します．このため，できるだけ少ない入力を持つ加算器で実装することが必要です．

Sobelフィルタでは，ピクセル・クロックがシステム・クロックの512倍遅くなり，計算時間が十分にあります．そこで，**図7**に示すように，繰り返し計算にしました．V行列とH行列に関する掛け算を（0，0）の要素から実行して足し合わせ，その結果をフィードバックしています．

このように掛け算の結果を加算して累積していく処理は積和演算と呼ばれます．結果は絶対値をとるため，最後に絶対値を計算し，出力しています．

リスト1に積和演算と絶対値計算のVHDLコードを示します．

リスト1の①では，8ビットでの計算の精度を上げるため，1ビット分，符号ビットを拡張しています．これで，純粋な8ビットの計算が可能になっています．

リスト1の②は乗算器です．Spartan-3Eのように乗算器を持つFPGAをターゲットにする場合，VHDLでは“*”を使って掛け算を記述すれば，ハード・マクロの乗算器を利用するように論理合成・配置配線されます．

リスト1の③は掛け算の結果と，前回の積和演算の結果とを加算しています．加算の結果，オーバフローするときには（つまり，0xFFFF＋0xFFFFのように16ビット目にけた上がりが発生している場合），0xFFFFに値を強制的に変更しています．また，アンダフローのとき（つまり，0x0000－0xFFFFのような場合）には0x0000に値を強制的に変更しています．

リスト1　積和演算と絶対値計算のVHDLコード

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity MAC_sobel is
    Port ( clk : in std_logic;
           rst : in std_logic;
           DinA : in std_logic_vector(7 downto 0);    ← 行列Iの要素入力
           DinB : in std_logic_vector(7 downto 0);    ← 行列VまたはHの要素入力
           Dout : out std_logic_vector(7 downto 0));  ← MAC演算の結果
end MAC_sobel;

architecture Behavioral of MAC_sobel is
      signal A, B : std_logic_vector(8 downto 0);
      signal MULT_o, ACC, ACC_act  : std_logic_vector(16 downto 0);
begin
A <= '0' & DinA(7 downto 0);         ┐①符号のために1ビット拡張する
B <= DinB(7) & DinB(7 downto 0);     ┘

process (A, B)                                   ┐
variable aux: std_logic_vector(17 downto 0);     │
begin                                            │②入力された要素を掛け算する
      aux := A*B;                                │
      MULT_o <= aux (16 downto 0);               │
end process;                                     ┘

process (MULT_o, ACC_act)                        ┐
variable aux: std_logic_vector(17 downto 0);     │
begin                                            │
      aux := (MULT_o(16) & MULT_o) + (ACC_act(16) & ACC_act);
      if aux(17)='0' and aux(16)='1' then   --overflow
              ACC(16)<='0';                      │
              ACC(15 downto 0)<=(others=>'1');   │③フィードバックされた値と掛け算の結果を足し合わせる
      elsif aux(17)='1' and aux(16)='0' then         --underflow
              ACC(16)<='1';                      │
              ACC(15 downto 0)<=(others=>'0');   │
      else                                       │
              ACC(16 downto 0)<=aux(16 downto 0);│
      end if;                                    │
end process;                                     ┘

process(clk, rst)                                ┐
begin                                            │
      if rst='1' then                            │
              ACC_act <= (others => '0');        │④MACの結果をラッチする
      elsif clk'event and clk='1' then           │
              ACC_act<=ACC;                      │
      end if;                                    │
end process;                                     ┘

process(ACC_act)                                 ┐
variable aux: std_logic_vector(16 downto 0);     │
begin                                            │
      aux := ACC_act;                            │
      if aux(16)='0'then                         │
              Dout <= aux(9 downto 2);   ← MSBの0のときはそのまま
      else                                       │⑤MACの結果の絶対値を取る
              Dout <= (not aux(9 downto 2)) + 1; ← MSBが1のときは2の補数の逆演算をする．
      end if;                                    │  つまり，正の値に変換
end process;                                     │
                                                 │
end Behavioral;                                  ┘
```

リスト1の④で，積和演算の結果を保持し，**リスト1**の⑤の絶対値の演算で，正の値に変更しています．

● データ転送モジュールはフリーのIPコアを活用

画像処理モジュールにおける計算結果は，データ転送モジュールが用意するバッファに一度書き込まれます．このバッファから順次データが読み出され，シリアル・ポートから表示ソフトウェアへと転送されていきます．

シリアル通信モジュールには，フリーで利用できるシリアル通信IPコアを用いました．インターネット上のOpenCoresのサイト(p.93のコラム「OpenCoresとWISHBONEインターフェース」を参照)にある「MiniUART」です(**リスト2**)．シリアル・ポート側の端子としてRXとTXのみを持つ非常に単純な機能のコアです．

内部回路へのインターフェースは，WISHBONEと呼ばれるOpenCoresで共通に用いられている標準バスになっています．

MiniUARTコアは，BR_CLK_Iという信号にクロックを供給します．今回は50MHzを入力しました．通信レートを決めるためには，定数BRDIVISORを使います．今回は130に設定し，19200bpsの通信としています．

● 表示ソフトウェアをC#で記述

表示ソフトウェアはC#で記述しました．PictureBoxコントロールに受信したピクセル・データをSetPixelメソッドで表示します．

シリアル・ポートの制御にはSerialPortコントロールを取り込み，その受信イベントで，ヘッダ，イメージ・サイズ，ピクセル値，フッタを順次，受信するようにします．

リスト2　MiniUARTのインスタンス（UART_controller.vhd）

```
COMPONENT UART
GENERIC(BRDIVISOR: INTEGER range 0 to 65535 := 130); -- Baud rate divisor
PORT (
      -- Wishbone signals
      WB_CLK_I : in  std_logic;  -- clock
      WB_RST_I : in  std_logic;  -- Reset input
      WB_ADR_I : in  std_logic_vector(1 downto 0); -- Adress bus
      WB_DAT_I : in  std_logic_vector(7 downto 0); -- DataIn Bus
      WB_DAT_O : out std_logic_vector(7 downto 0); -- DataOut Bus
      WB_WE_I  : in  std_logic;  -- Write Enable
      WB_STB_I : in  std_logic;  -- Strobe
      WB_ACK_O : out std_logic;  -- Acknowledge
      -- process signals
      IntTx_O  : out std_logic;  -- Transmit interrupt: indicate waiting for Byte
      IntRx_O  : out std_logic;  -- Receive interrupt: indicate Byte received
      BR_Clk_I : in  std_logic;  -- Clock used for Transmit/Receive
      TxD_PAD_O: out std_logic;  -- Tx RS232 Line
      RxD_PAD_I: in  std_logic); -- Rx RS232 Line
END COMPONENT;
```

コラム　I²C-bus

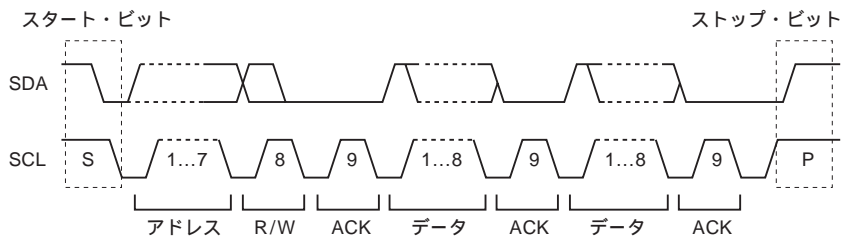
I²C-bus（Inter Integrated Circuit bus）は，オランダPhilips Semiconductors社（現在はNXP Semicnductors社）により提唱されたバスです．従来，ライセンス契約とロイヤリティの支払いが必要でしたが，2004年8月から不要になりました．

I²C-busは，多くのLSIで内部レジスタなどの書き換え動作やA-D/D-Aコンバータのデータ読み書きインターフェースに採用しています．周辺制御用バスの業界標準の一つです．

バスとしての特徴は，1本のデータ線（SDA）と1本のクロック線（SCL）の2本だけでデータ転送が可能なプロトコルを定義しているところです．最大4.3Mビット/sでの転送が可能な仕様が公開されています．

I²Cでは，マスタとスレーブという2種類のデバイスを定義しています．マスタがスレーブに読み書きの要求を送信します．複数のスレーブがバスに接続されている状況では，バス・サイクルのアドレス・フィールドでスレーブを特定します．本稿で用いている転送方式は，7ビット・アドレス・フォーマットと呼ばれるもので，**図A**に示すようなタイミングでデータ転送を実行します．

SCLが“H”の時，SDAの立ち下がりエッジにおいてスレーブはバス・サイクルが始まったと認識します（スタート・ビット）．そこから，SCLの立ち上がりエッジに同期して，アドレス，イネーブル（R/W），データなどを受け取り，ACKを送信します．ACKはスレーブがSDAを“L”にすることで行われます．このときデータはスレーブで正しく処理されたことを示しています．このデータとACKを繰り返すことで8ビットずつの転送が行われます．SCLが“H”の時，SDAの立ち上がりエッジが来るタイミングでバス・サイクルの終了となります（ストップ・ビット）．

図A　I²Cのデータ転送サイクル
1本のデータ線（SDA）と1本のクロック線（SCL）の2本だけでデータ転送を行う．

まず，Connect Serial Portボタンをクリックし，シリアル・ポートの設定をします．ここでのシリアル・ポートは**図8**に示すような設定になっています．

ボードの取り込みボタンを押すと，シリアル・ポートからのデータを受け取り，ウィンドウに表示します．シリアル・ポートは一度開かれると開いたままになるので，ハードウェアからの0xFF，0x00…というヘッダ部分を受け取ると表示部が再起動され，画像が変わるようになっています．

図8　表示ソフトウェア
今回は19200bpsで通信する．

3．性能評価

● エッジ検出の効果

Sobelフィルタの効果を見るために，フィルタなしとフィルタありの画像を比べてみました．

フィルタなしの場合には，ハードウェア論理自体の変更は必要ありません．画像処理モジュールのROMに格納されている係数行列V(sobel_coefV.vhd)の中心の値を4，ほかを0と設定し，係数行列H(sobel_coefH.vhd)の要素値をすべて0とすることで，カメラからの直接画像と等価なものが得られます．

図9に10ユーロ札の左下に印刷されている10という文字を読み取り，エッジ検出している画像を示します．エッジがきれいに検出されているのが分かります．

コラム　OpenCoresとWISHBONEインターフェース

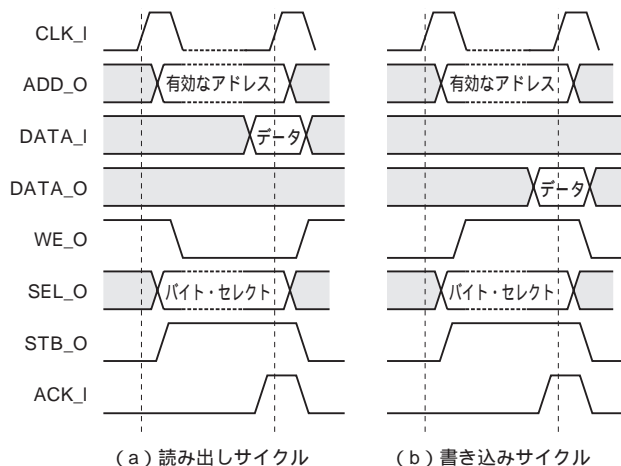
今や，多くの汎用IPコアはインターネットで入手できます．

OpenCoresのサイト(http://www.opencores.org/)では，最近の高周波で動作する回路や特殊用途向けのようなものを除き，USBやPCIといった汎用的な回路のIPコアをダウンロードして使うことができます．

IPコアはHDLソース・コードで提供されているので，シミュレーションなどもツールを選ばず可能になっています．ライセンスはGPL(GNU Public License)のため，完全にフリーのものがほとんどです．

一部のIPコアは商用では使えませんが，研究用途なら使えるライセンス体系のものもあります．

OpenCoresのIPコアの内部バス・インターフェースはすべてWISHBONEと呼ばれるバス規格で統一されています．このバスは素直で単純なプロトコルを定義しています．バスはバス・マスタによってドライブされ，バス・スレーブが反応します．マスタのバスにはクロック(CLK_I)，アドレス(ADR_O)，入力データ(DATA_I)，出力データ(DATA_O)，書き込みイネーブル(WE_O)，バイト・セレクタ(SEL_O)，ストローブ(STB_O)，アクノリッジ(ACK_I)があります．

図Bにマスタがスレーブに対して，シングル・データの読み出しと書き込みをするタイミング・チャートを示します．マスタはストローブとアドレス，そして，バイト・セレクタをドライブして，スレーブが反応するのを待ちます．データが読み出されるか，書き込みが完了すると，そのタイミングでスレーブはアクノリッジをアサートします．

図B　WISHBONEインターフェースの読み書きサイクル
WISHBONEは，OpenCoresの標準バスである．素直で単純なプロトコルであることが分かる．

（a）フィルタなし

（b）Sobelフィルタあり

図9　エッジ検出処理の結果
10ユーロ札の左下に印刷されている10という文字を読み取り，エッジ検出している．

表1　コンパイル結果

	スライス数	clk_iの周波数
スライスで実装	475	13.942ns
乗算器を使って実装	427	14.084ns

● ハードウェア乗算器の効果

Spartan-3Eの特徴の一つとして，ハード・マクロの乗算器があります．今回設計した回路を，乗算器を使用した場合と，乗算回路を論理ブロックで構成した場合とについて比較してみました．

表1に，スライスのみで乗算器を実装したときとハード・マクロの乗算器を用いて実装したときのハードウェア使用量と周波数を示します．論理ブロックを用いて乗算回路を構成した場合に比べ，ハード・マクロの乗算器を用いた場合は，ハードウェア規模の縮小が可能であることが示されています．従って，Spartan-3Eでは，画像処理や信号処理を効率良く実装可能なことが分かります．

やまぎわ・しんいち
José Germano
ポルトガルINESC-ID（Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento）/ Technical University of Lisbon

<筆者プロフィール>
山際伸一．ポルトガルの研究所INESC-IDのシニア研究員．博士（工学）．並列分散処理，特にクラスタ計算機の超高速ネットワーク・ハードウェアを専門とする．最近はGPUを使った高性能計算システムに関する研究を中心に多数の研究成果を発表している．CQ出版社「FPGAボードで学ぶ論理回路設計」をはじめとする書籍がある．
José Germano．リスボン工科大学電子工学科博士課程に在籍．マルチメディア・システムやLab-on-chip技術の経験を積み，現在は，DNA検出のための磁気抵抗バイオチップの研究・開発に邁進する．若きポルトガルのホープで，LSIおよび基板の設計・実装からFPGA内部回路設計まで，一切の実装関連技術をこなす．